

\*\* additional items to be added to site map

# All Components

* Identify all the data: states and props
* Identify the flow of data
* Identify all functions and handlers
* Identify all components
* Modify all components to make smaller components
* Add names to component elements that make sense for form input
* Add client-side and server-side validation
* Modify all components to make functional components and make parent state component
* Add lifecycle methods to all classes
* Analyze all lifecycle methods
* Add tests to all functions
* Send Email for actions needed for Donations & Pickups

Let’s transform all these tasks to user-value statements in Trello cards.

Try to reword the tasks in the following format:

As a…

I want…

So that…

For example

As a donor

I want to save profile information

So that I don’t have to repeat it each time I publish an item

# All Components

* Identify all the data: states and props
* Identify the flow of data
* Identify all functions and handlers
* Identify all components
* Modify all components to make smaller components
* Add names to component elements that make sense for form input
* Add client-side and server-side validation
* Modify all components to make functional components and make parent state component
* Add lifecycle methods to all classes
* Analyze all lifecycle methods
* Add tests to all functions

# Navbar

## Mobile Responsive

* ~~Add icon for toggle menu button~~
* ~~Move Donate to top, make logo smaller~~
* ~~For ipad, remove toggle, make logo smaller and possibly links smaller~~
* ~~Move keyword search and city/zip search to shop when collapse~~

# Footer

* Add US copyright symbol and year
* Add logo flower

# Shop

## Pending User Actions

* ~~Show actions needed for Donations & Pickups~~

## Test

* ~~Add unit tests for search conditions~~

# Sign Up Modal

## Validate

* ~~Validate all user data~~
* ~~Implement UI messages for user data~~
* ~~Redo passport and user authentication using local and google.~~
* ~~Understand every bit of code, comment all code.~~

# Login Modal

## Validate

* ~~Validate all user data~~
* ~~Implement UI messages for user data~~
* ~~Redo passport and user authentication using local and google.~~
* ~~Understand every bit of code, comment all code.~~

# Donate Form

## File Upload

* ~~Add conditional for file upload if dev or production~~
* ~~Add streaming to display picture~~
* ~~Upload file upon click Donate~~
* ~~Add disclosure for donation retention~~
* ~~Picture and input text scanned for hate and pornography~~

# I’m interested Modal

* ~~Create Mockup~~
* ~~Implement~~
* Send Email for actions needed for Donations & Pickups

# My Donations

* ~~Create Mockup~~
* ~~Implement~~

# My Pickups

* ~~Create Mockup~~
* ~~Implement~~

# User Profile

* ~~Create Mockup~~
* ~~Implement~~

# About US

* ~~Create Mockup~~
* ~~Implement~~

## FAQ

* ~~Add how to use steps~~

# **Data Models**

## **Document**

* **data models**
* **use cases**

# **Redux**

* **Add Redux: reducers, actions and store.**
* **Use Redux to feed stateful components**

# **Testing**

## **Hardware**

* **Desktop**
* **Laptop**
* **Ipad**
* **mobile**

# Legal

* Obtain US copyright
* ~~Security disclosure~~
* ~~Validate picture for no hate or pornography~~
* ~~Validate user input for no hate or pornography~~
* ~~Register with AWS for storage space~~
* Register with Mongodb (Mlab) for storage space
* ~~Domain Name~~
* ~~Deploy Domain~~

JEST – unit test framework

Cucumber

Make a jest branch

Figure out what order of priority

Make cards with business

Estimate stories in terms of complexity in points

Adding an icon in relation to the pointer

What is Test Driven Development? Coding of features and tests go hand in hand.

1. Write a unit test
2. Run the test. See it fail.
3. Write the feature code to pass the test
4. Refactor the code

What is Behavior Driven Development (BDD)?

* A variation of TDD that tests for user scenarios
* Given, when, then
* Given notes, when deleting, then remove a note
* Bdd consists of scenarios/specifications

Jest + Enzyme = TDD with React

* Jest (Facebook) – test JavaScript, includes snapshot testing, coverage and mocking (check if something was called).
* Enzyme (Airbnb) – Allow us to work with specific components. Testing of manipulating React components and DOM behavior. Ability to find specific items in component itself.

**Installing enzyme**

Install enzyme with jest cli: npm install enzyme jest-cli@20.0.4 --save-dev

Install react bootstrap: npm install react-bootstrap –save

Run tests: npm run test (will run test with test.js extension)

If problems

remove node folder: rm -rf node\_modules

To reinstall node modules: npm i

Install react test render: npm i react-test-render --save-dev

Install adapter for enzyme and react: npm i enzyme-adapter-react-16 --save-dev

**Testing with jest**

Shallow – helper function from enzyme, will only render out JSX, no child components.

it('renders correctly', () => {

expect(app).toMatchSnapshot();

})

Important: Enzyme Adapter

Section 2, Lecture 8

Please keep this in mind!

In order to use the most current version of React > 16, we now need to install "enzyme adapters" to provide full compatibility with React.

There's two options to take. The first is quicker, the second is recommended:

1. You can use the same versions for the node\_modules libraries in the package.json files of the original repo:

Or you can:

2. Install the adapter for enzyme and react:

npm i enzyme-adapter-react-16 --save-dev

Next, add a src/tempPolyfills.js file to create the global request animation frame function that React now depends on.

src/tempPolyfills.js should contain the following contents:

const requestAnimationFrame = global.requestAnimationFrame = callback => {

setTimeout(callback, 0);

}

export default requestAnimationFrame;

Finally, add a src/setupTests.js file to configure the enzmye adapter for our tests. The disableLifecyleMethods portion is needed to allow us to modify props through different tests.

src/setupTests.js should contain the following contents:

import requestAnimationFrame from './tempPolyfills';

import { configure } from 'enzyme';

import Adapter from 'enzyme-adapter-react-16';

configure({ adapter: new Adapter(), disableLifecycleMethods: true });

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Let's keep coding!

// check if component renders correctly

// to create a jest unit test, we use jest method 'it' function

// it takes two args, test desc and test

it('renders correctly', () => {

// expect jest method, allows us to check if parts of this app component meet certain conditions

// testing conditions are asserted by chaining other jest methods after the expect method

// toMatchSnapshot is a key method of snapshots. Snapshots keep a recorded history of your react components,

// everytime a change is made in the application snapshot will update and make sure previous components

// in testing files match your new one

// using snapshot we can test the rendered JSX as a whole

expect(app).toMatchSnapshot();

});

// once me make changes to the rendered component of App.js, the test will fail,

// because the rendered component is different than the snapshot.

// We can then confirm the change and by pressing u to update the snapshot

it('initializes the `state` with an empty list of gifts', () => {

expect(app.state().gifts).toEqual([]);

});

it('adds a new gift to `state` when clicking the `add gift` button', () => {

// find add gift button and click

// find method lets us find child or inner components by JSX tags or by className

app.find('.btn-add').simulate('click');

expect(app.state().gifts).toEqual([{id: 1}]);

});

it('adds a new gift to the rendered list when clicking the `add gift` button', () => {

app.find('.btn-add').simulate('click');

expect(app.find('.gift-list').children().length).toEqual(2);

})

* With BDD it’s a common practice to group a bunch of tests together.
* Use Jest Global **describe** method to group tests together,
* \*\*\* Snapshot becomes obsolete
* Press u to update
* Press a to rerun tests

Section Summary | GiftGiver and Test Driven Development

Section 2, Lecture 19

Section Summary | GiftGiver and Test Driven Development

Excellent work on completing GiftGiver in the approach of Test and Behavior Driven Development! Here’s an overview of the main concepts that appeared in this section:

Test Driven Development is an approach to coding in which you write tests before features. The approach generally follows these steps.

First: write a single unit test to check on a part of the app’s behavior.

Second: observe the test failing.

Third: Write the code to pass the test.

Finally, refactor the code to reduce duplication and lines.

A unit test checks the smallest possible part of an application for proper behavior.

Jest is a test runner made by Facebook to test JavaScript environments and is most frequently used to test React applications.

Snapshot testing in Jest keeps a recorded history of testing files in order to ensure that previous snapshots match current snapshots that generate when the application changes.

Enzyme is a testing utility library made by Airbnb that allow us to interact with React components.

Behavior Driven Development is a variation of test driven development that hones in on user scenarios and behaviors that affect the app state.

Given, when, and then/and represent key words in the BDD structure and help document the user scenarios for the application.

Again, great job on completing GiftGiver. Next, we’ll learn more advanced techniques for behavior driven development that will guide us towards creating more complex applications. So let’s keep coding!

Loot Check Application

Install test dependencies: npm i react react-dom react-test-renderer jest-cli@20.0.4 enzyme --save-dev

npm i enzyme-adapter-react-16 --save-dev

Install production dependencies: npm i redux react-redux sfcookies --save

# Redux – managing data in React

* A huge app has a complex UI
* Dozens of components pass data
* What if a child need to update a parent?

Redux

* The **store** is one data object for the app
* **Reducers** update the store
* **Actions** activate reducers
* Any child component can update the store, which can be reflected on sibling or parent components
* This provides a very convenient way to manage the data in the application

Store

* { stacks cards }

Reducers

* Knows how to update the store
* Each reducer has rules

Actions

* Messages to reducers to tell what to do, like add new card and data explaining what kind of card it is

Section Summary | LootCheck and Behavior Driven Development | Part One

Section 3, Lecture 33

**Section Summary | Lootcheck and Behavior Driven Development | Part One**

Excellent work on completing the first part of Lootcheck with Behavior Driven Driven Development! In this section, we covered a lot of ground and saw more in-depth techniques for test and behavior driven development. Let’s highlight them:

* Testing **Redux Actions and Reducers**involves testing the output of pure functions - functions that return the same output based on the same input. That means we can build the expected output for an input, and test the action/reducer based on that input and output.
* The **Redux Store**is one giant object that represents the entire state of an application.
* Redux Reducers provide guidline on how to update the store based on different redux actions.
* Redux Actions are objects with packets of data a specific type that identify it to the redux reducer.
* Action Types are SCREAM\_CASED strings that uniquely identify a redux action.
* Action Creators are functions that create and return redux action.
* We can provide mock functions with **jest.fn()** to the props interface of a component to check whether or not redux action creators are dispatched.
* **Cookies**allow us to store information on the user’s browser with a unique string.

# Async Actions, Middleware & Redux Thunk

Async Actions

* Actions in redux that allow for asynchronous behavior
* An API request shouldn’t block the application
* The API request should happen in the background

How do async actions work?

* Use **middleware**
* Add a layer between the async code and the store
* **Redux-thunk** – use to return an API Promise, rather than the normal action object

Install Redux Thunk: npm i redux-thunk –save

npm i redux-mock-store fetch-mock --save-dev

Section Summary | Lootheck and Behavior Driven Development | Part Two

Section 4, Lecture 45

**Section Summary | Lootcheck and Behavior Driven Development | Part Two**

Congratulations on completing Lootcheck - the second and final application of this course on Test and Behavior Driven Development with React. In this section, we covered a lot of important concepts. So let’s go over them:

* Asynchronous code does not block the application from running. It runs in the background and allows other features/aspects of the application to occur.
* Asynchronous actions in redux return more than plain objects. They can return functions and promises.
* Middleware adds a layer between the redux store and code like an asynchronous fetch request to an API.
* Redux Thunk is a piece of middleware that allows for actions to return JavaScript promises.
* Fetch-Mock mocks the native fetch() method of JavaScript in the testing environment by stubbing http endpoints.
* The redux combineReducers() combines more than one reducer into one overall reducer for an application.

In addition to covering a lot of new ground with these more advanced concepts, we gained more practice with Test and Behavior Driven Development while building Lootcheck. By completing this application, you’ve not only finished this section, but also the course. Excellent work! With the power of Test and Behavior Driven Development, you will create robust React and JavaScript applications that will be highly efficient and reliable.

Stick around for the final conclusion section. There’s more to take note of and some bonus content!